PROTOTIPE KOMPRESI LOSSLESS AUDIO CODEC MENGGUNAKAN ENTROPY ENCODING

Andreas Soegandi

Jurusan Teknik Informatika, Fakultas Ilmu Komputer, Bina Nusantara University
Jln. K.H. Syahdan No. 9, Palmerah, Jakarta Barat 11480
soegandi@binus.edu

ABSTRACT

The purpose of this study was to perform lossless compression on the uncompress audio file audio to minimize file size without reducing the quality. The application is developed using the entropy encoding compression method with rice coding technique. For the result, the compression ratio is good enough and easy to be developed because the algorithm is quite simple.

Keywords: *compression, audio, entropy, lossless, encoding*

ABSTRAK

Tujuan dari penelitian ini adalah untuk melakukan kompresi secara lossless pada file audio agar dapat memperkecil ukuran file tanpa mengurangi kualitas. Pembuatan aplikasi untuk melakukan kompresi didasarkan pada metode entropy encoding dengan teknik rice coding. Dari hasil penelitian didapatkan bahwa rasio kompresi cukup baik dan mudah untuk dikembangkan karena algoritma yang digunakan cukup sederhana.

Kata kunci: *Kompresi, audio, entropy, lossless, encoding*

PENDAHULUAN

Pada masa informasi sekarang ini, bentuk – bentuk data yang digunakan bukan hanya berupa teks, tetapi berupa elemen multimedia seperti gambar, animasi, suara dan video. Data dan informasi dalam bentuk multimedia memang lebih menarik dibandingkan data dan informasi dalam bentuk teks, namun ukuran file elemen multimedia ini jauh lebih besar dibandingkan dengan teks.

Suara atau audio merupakan salah satu elemen multimedia yang mempunyai ukuran cukup besar pada kualitas terbaiknya. Ada 2 bentuk format file audio yang digunakan yaitu *uncompressed* audio file (.wav) dan *compressed* audio file. Besar file *uncompressed* audio file sama dengan besar kapasitas audio CD sedangkan besarnya file *compressed* audio file lebih kecil daripada audio CD namun biasanya kualitas suaranya jauh dibawah audio CD karena menggunakan metode kompresi *lossy*.

Kebanyakan kompresi yang ada adalah kompresi lossy karena penurunan kualitas tidak terlalu dirasakan oleh orang awam, namun untuk sebagian orang, penurunan kualitas ini sangat terasa. Hal tersebut yang menjadi latar belakang penelitian ini yaitu mengembangkan suatu aplikasi yang mengimplementasikan algoritma kompresi dengan tidak mengurangi kualitas. Kompresi seperti ini dikenal sebagai *lossless compression*.

Kompresi Data

Dalam ilmu komputer, kompresi data adalah sebuah cara untuk mengubah data sehingga hanya memerlukan ruangan atau kurang waktu pertukaran lebih kecil. Kompresi data ada 2 macam, yaitu *lossy* dan *lossless compression*. Pada *lossy compression* akan ada penghapusan data sehingga rasio kompresi yang dihasilkan tinggi tetapi kualitasnya menjadi lebih rendah dibandingkan file aslinya, sedangkan pada *lossless compression* tidak ada penghapusan data sehingga hasil rasio kompresi yang dihasilkan tidak sebaik *lossy compression*, namun kualitasnya sama dengan file aslinya.

Ada beberapa algoritma untuk mengkompresi dengan teknik *lossless compression* seperti *rice coding* dan *huffman coding*. Pada penelitian ini algoritma yang akan digunakan adalah *rice coding* karena *rice coding* tidak bergantung pada probabilitas data yang akan dikodekan berbeda dengan *huffman coding* yang bergantung pada probabilitas data yang akan dikodekan. Jika dihitung kemungkinan kombinasi angka yang akan muncul pada file audio .wav dengan bitrate 16-bit adalah 65.536 angka, dimana besar angka tersebut berkisar antara 0 – 65.535 (dalam *hexadecimal* antara 0000 – FFFF). Banyaknya kombinasi angka yang dapat muncul pada perhitungan ini akan membuat algoritma *huffman coding* lebih buruk dalam mengkodekan data daripada algoritma *rice coding* (Morken, 2007:113-116).

Format Data Audio

Dalam menyimpan audio dalam media elektronik, digunakan standar format file audio milik Microsoft dan IBM yang disebut *waveform audio format* (.wav atau *wave*). File audio .wav kompatibel dengan sistem operasi Windows dan Macintosh. File audio .wav merupakan variasi metode format *bitstream* .riff untuk menyimpan data ke dalam potongan – potongan data / *chunks*. Format *bitstream* .riff berfungsi sebagai pembungkus untuk berbagai macam *audio compression codec*. File audio .wav biasanya berisikan *uncompressed* audio dalam format PCM (*Pulse Code Modulation*) (McGloughlin, 2001:268,277).

Entropy Encoding

Sebuah *entropy encoding* adalah sebuah skema *coding* yang memberikan kode-kode ke simbol-simbol untuk menyamakan dengan panjang kode dengan probabilitas dari simbol-simbol. Teknik *entropy encoding* yang umum adalah *huffman coding, arithmetic coding, elias gamma coding, fibonacci coding,* dan *rice coding* (Whiters, 1996:2).

Rice Coding

Rice coding ditemukan oleh Robert F. Rice dan merupakan salah satu teknik *entropy encoding*. *Rice coding* atau biasa disebut juga *rice packing* adalah suatu cara untuk mempresentasikan angka-angka kecil, sementara tetap mempertahankan kemampuan untuk mengetahui angka berikutnya (Ashland:2000).

Urutan bit dalam *rice coding* adalah
- Sign bit, 1 untuk angka positif dan 0 untuk angka negatif
- Bit 0 sebanyak n / (2^k)
- *Terminating* bit 1
- Bit paling kanan sebanyak k dari integer n

Dimana n adalah angka yang akan diproses, dan k adalah parameter. Untuk menghitung parameter bisa digunakan rumus log(n) / log(2). Pada kenyataannya parameter tidak bisa dihitung untuk satu angka saja melainkan satu parameter digunakan untuk beberapa angka.

Contoh *rice coding* untuk angka 157 dengan parameter 6 :
1. *sign bit* → [1]
2. n / (2^k) = 157 / 2^5 = 2 → [00]
3. *Terminating* bit → [1]
4. 6 bit paling kanan → [011101]
5. Gabungkan langkah 1 - 4 → [1][00][1][011101]
6. Maka *rice code* untuk angka 157 adalah 1001011101

Untuk mengembalikan urutan bit ke dalam desimal digunakan langkah :
1. Baca *sign bit*, 1 untuk positif dan 0 untuk negatif
2. Baca bit 0 sampai menemukan bit 1, dan simpan banyaknya bit 0 yang dibaca
3. Baca bit dari kanan sebanyak nilai parameter
4. Gabungkan bit-bit yang didapat dari Langkah 3 dengan *binary* dari banyaknya bit 0 yang didapat dari Langkah 2. Ubah gabungan bit tersebut ke bentuk desimal dan kalikan dengan *sign bit*. Kalikan dengan satu untuk *sign bit* 1 dan kalikan dengan -1 untuk *sign bit* 0.

Contoh *rice decoding* :

Rice code : 1001011101, Parameter: 3
1. 1001011101, *sign bit* = 1 maka positif.
2. 1001011101, bit 0 yang terbaca sebanyak 2 → [10]
3. 1001011101, *terminating bit*
4. 1001011101, baca 6 bit dari kanan → [011101]
5. Gabungan bit → [10][011101], desimalnya adalah 157

METODE

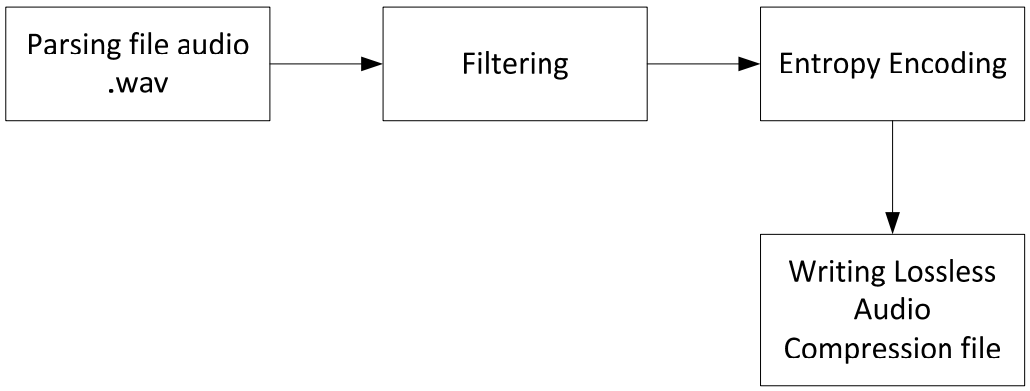
Metode penelitian yang digunakan adalah dengan melakukan studi pustaka terhadap teori yang berkaitan dengan proses kompresi, *encoding* dan *decoding*. Analisis dilakukan dengan membandingkan hasil kompresi dari teknik – teknik yang ada. Perancangan dilakukan dengan menggunakan diagram blok dan diagram alir (*flowchart*) untuk menyusun algoritma yang akan digunakan.

Pengujian dilakukan dengan membandingkan ukuran file hasil kompresi, rasio kompresi, waktu kompresi dan waktu dekompresi. Selain itu, juga dilakukan perbandingan hasil kompresi yang dilakukan oleh aplikasi lain yang memiliki fungsi sama untuk mengkompresi file audio *.wav* tanpa mengurangi kualitas suara (*lossless*).

HASIL DAN PEMBAHASAN

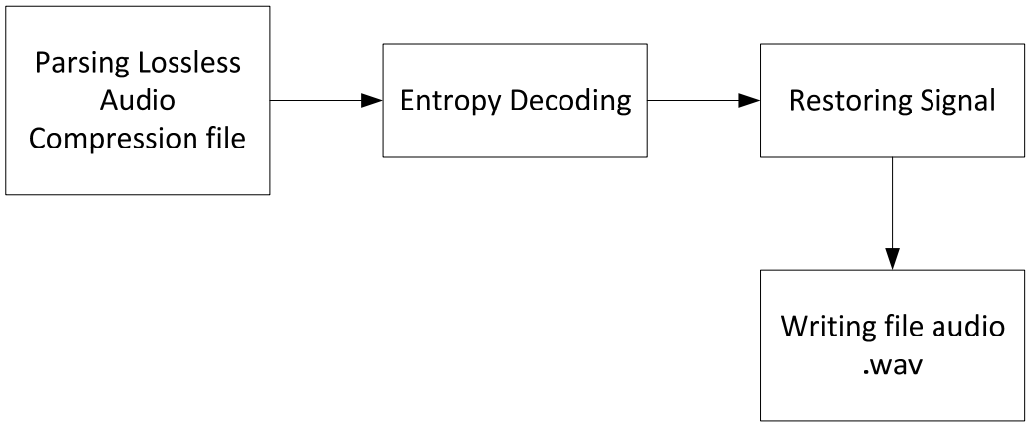
Analisis Proses dan Algoritma

Dalam proses kompresi, file audio *.wav* yang akan dikompres akan divalidasi terlebih dahulu pada proses parsing (Gambar 1). Setelah melewati proses validasi, file tersebut akan dipecah menjadi dua sinyal mono dan diambil nilai sampelnya. Nilai sampel yang telah diambil tersebut menjadi input untuk proses filtering, dimana pada proses ini nilai sampel tersebut akan dikecilkan nilainya dengan menggunakan algoritma prediksi. Angka yang lebih kecil tersebut dinamakan *residual error*. *Residual error* tersebut akan dimasukkan ke proses *entropy encoding* dimana pada proses ini, nilai *residual error* akan dikodekan sehingga pemakaian bit menjadi lebih sedikit. Dalam melakukan pengkodean, digunakan algoritma *rice coding*.



Gambar 1. Proses kompresi (*encoding*)

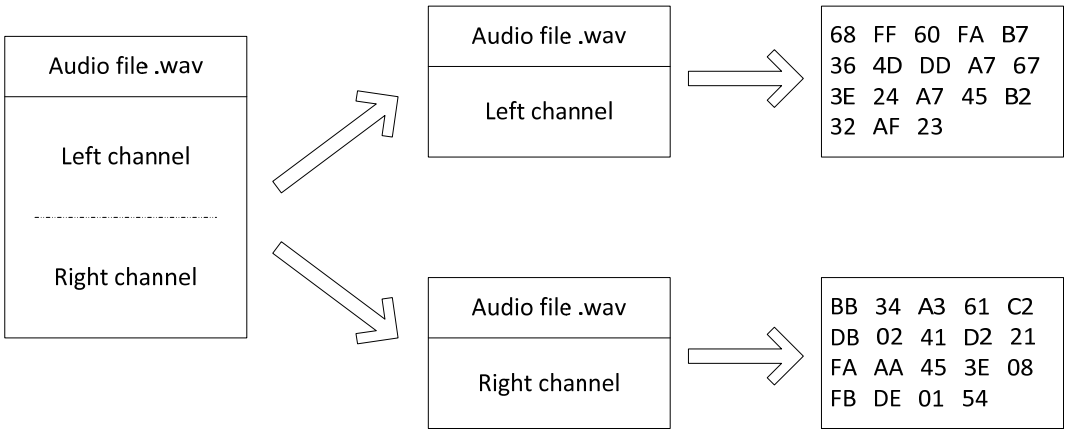
Dalam proses dekompresi, file *Lossless Audio Compression* (LAC) yang akan didekompresi diambil datanya dan dimasukkan ke proses *entropy decoding* (Gambar 2). Keluaran dari proses tersebut adalah *residual error*. Nilai *residual error* kemudian dimasukkan ke proses pengembalian sinyal (*signal restoring*). Proses ini menggunakan *residual error* untuk merekonstruksi nilai sampel asli.



Gambar 2. Proses dekompresi (*decoding*)

Perancangan Proses *Encoding*

- *Parsing* .wav file
  Di dalam proses ini, pada file input akan dilakukan validasi dan akan dilakukan parsing jika lolos dari validasi. Validasi dilakukan pada struktur file input. Jika struktur file input tidak sama dengan standar struktur file audio .*wav*, maka aplikasi akan selesai (tidak akan dilanjutkan ke tahap berikutnya). Jika struktur file input sesuai maka aplikasi akan melanjutkan ke proses *parsing*. Proses *parsing* akan membagi sinyal stereo yang ada di dalam file audio .*wav* menjadi sinyal mono. Setelah dibagi setiap sinyal mono itu akan diambil nilai-nilai sampelnya. Pembacaan pada file audio .*wav* memakai cara binary.



Gambar 3. Proses *parsing* file audio .wav

- *Filtering*
  Proses *filtering* akan memproses nilai-nilai sampel yang telah didapatkan pada proses *parsing*. Nilai-nilai sampel tersebut mempunyai nilai angka yang besar, proses *filtering* akan mengolah nilai-nilai sampel menjadi nilai angka yang lebih kecil, sehingga proses *entropy encoding* bisa lebih efektif mengolah nilai-nilai tersebut. Teknik yang digunakan untuk melakukannya adalah

prediksi. Algoritmanya adalah nilai sampel berikutnya diprediksi menggunakan nilai sampel sebelumnya.

$$p_n = x_{n-1}$$

$$e_n = x_n - p_n$$

Dimana p adalah nilai prediksi, x adalah nilai sampel, e adalah nilai *residual error*, dan n adalah posisi sampel sekarang. Yang disimpan dalam proses *filtering* adalah nilai residual error untuk setiap nilai sampelnya.

Pada Tabel 1 dapat dilihat contoh penghitungan *residual error* yang dijalankan pada *filtering* nilai sampel.

Tabel 1. Contoh Perhitungan *Residual Error* Pada Proses *Filtering*

#	Nilai Sampel Asli (x)	Nilai Prediksi (p)	Nilai Residual Error (e)
1	1945	0	1945
2	1877	1945	-68
3	1723	1877	-154
4	1552	1723	-171
5	1309	1552	-243
6	1024	1309	-285
7	814	1024	-210
8	513	814	-301
9	145	513	-368
10	50	145	-95
11	-214	50	-264
12	-456	-214	-242
13	-354	-456	102
14	-104	-354	250
15	-34	-104	70
16	24	-34	58
17	75	24	51
18	157	75	82
19	357	157	200
20	765	357	408

- *Entropy Encoding*
  Keluaran dari proses *filtering* yaitu *residual error*, selanjutnya dimasukkan ke proses *entropy encoding*. Proses ini menggunakan algoritma *rice coding*. Proses *entropy encoding* dimulai dari perhitungan parameter optimal untuk setiap nilai *residual error*. Setelah perhitungan selesai maka nilai *residual error* tersebut akan dikelompokkan sebesar 50 nilai setiap kelompoknya secara urut. Untuk setiap kelompok tersebut dihitung nilai parameter yang paling optimal. Cara menghitung nilai parameter yang optimal untuk kelompok tersebut adalah dengan cara menghitung nilai parameter yang optimal untuk semua nilai *residual error* kemudian diambil nilai yang paling besar dan dikurangi satu. Setelah perhitungan parameter barulah untuk setiap nilai *residual error* itu diproses dengan algoritma *rice coding*. Keluaran dari proses *entropy encoding* ini adalah *bitstream*

dari nilai-nilai *residual error* yang telah dikodekan dengan menggunakan algoritma *rice coding*. Hasilnya dapat dilihat pada Tabel 2.

Tabel 2. Contoh *Rice Coding* dengan Parameter 8

#	Nilai Residual Error	Rice Code
1	100	1101100100
2	-68	0101000100
3	-154	0110011010
4	-171	0110101011
5	-243	0111110011
6	-285	00100011101
7	-210	0111010010
8	-301	00100101101
9	-368	00101110000
10	-95	0101011111
11	-264	00100001000
12	-242	0111110010
13	102	1101100110
14	250	1111111010
15	70	1101000110
16	58	1100111010
17	51	1110110011
18	82	1101010010
19	200	1111001000
20	520	100100001000

Dari tabel *rice coding* dengan Parameter 8 didapatkan bahwa *bitstream* yang dihasilkan adalah 1101100100010100010001100110100110101011011111001100100011101011101001000100101 1010010111000001010111110010000100001111100101101100110111111101011010001101100 1110101110110011110101001011110010001001000010000.

- *Writing Lossless Audio Compression* (LAC) File
  Proses ini adalah proses penulisan file output. File output ini mempunyai struktur tersendiri yang dapat dilihat pada Tabel 3.

Tabel 3. Struktur File *Lossless Audio Compression* (LAC)

Nama Field	Panjang (Byte)	Keterangan
String ‘SLAC'	4	String LAC
String ‘FRMT'	4	
Panjang format Chunk	4	
Tipe format Chunk	2	
Jumlah Channel	2	Format Chunk
Rata - rata Sampel	4	
Rata - rata Byte	4	
Blockalign	2	
Nilai bits sampel	2	
String ‘METADATA'	8	
Jumlah Sampel tiap channel	4	
Jumlah Sampel channel kiri	4	Metadata Chunk
Jumlah Sampel channel kanan	4	
Besar blok	2	

<kosong>	2	Parameter Chunk
String ‘PRMT'	4	
Parameter	X	
String ‘DATA'	4	Data Chunk
Data	X	
String ‘END'	4	

Evaluasi

Evaluasi yang dilakukan pada *Lossless Audio Codec* (LAC) ini difokuskan pada ukuran file hasil kompresi, rasio kompresi, waktu kompresi dan waktu dekompresi. Selain itu, juga dilakukan perbandingan hasil kompresi yang dilakukan oleh aplikasi lain yakni *WavPack* dan *Free Lossless Audio Codec* (FLAC) yang memiliki fungsi sama untuk mengkompresi file audio .*wav* tanpa mengurangi kualitas suara (*lossless*). Pada evaluasi dilakukan analisa terhadap 5 jenis musik (*music genre*) yaitu musik klasik, jazz, R’n B, rock dan instrumental.

- Evaluasi pada musik klasik
  Dari hasil evaluasi pada musik klasik dapat diambil rata – rata perkiraan rasio kompresi pada tiap aplikasi kompresi adalah sebagai berikut :
  - LAC : 55 % - 63 %
  - WavPack : 36 % - 47 %
  - FLAC : 39 % - 51 %

- Evaluasi pada musik jazz
  Dari hasil evaluasi pada musik jazz dapat diambil rata – rata perkiraan rasio kompresi pada tiap aplikasi kompresi adalah sebagai berikut :
  - LAC : 67 % - 70 %
  - WavPack : 58 % - 61 %
  - FLAC : 60 % - 64 %

- Evaluasi pada musik R’nB
  Dari hasil evaluasi pada musik R’nB dapat diambil rata – rata perkiraan rasio kompresi pada tiap aplikasi kompresi adalah sebagai berikut :
  - LAC : 72 % - 79 %
  - WavPack : 61 % - 67 %
  - FLAC : 65 % - 72 %

- Evaluasi pada musik rock
  Dari hasil evaluasi pada musik rock dapat diambil rata – rata perkiraan rasio kompresi pada tiap aplikasi kompresi adalah sebagai berikut :
  - LAC : 79 % - 83 %
  - WavPack : 70 % - 75 %
  - FLAC : 74 % - 79 %

- Evaluasi pada musik instrumental
  Dari hasil evaluasi pada musik klasik dapat diambil rata – rata perkiraan rasio kompresi pada tiap aplikasi kompresi adalah sebagai berikut :
  - LAC : 63 % - 68 %
  - WavPack : 45 % - 51 %
  - FLAC : 48 % - 55 %

Dari kelima contoh jenis musik yang di evaluasi tampak hasil yang berbeda pada tiap jenisnya. Dari hasil evaluasi ini dapat disimpulkan bahwa perbedaan jenis musik *(musik genre)* mempengaruhi besar kompresi. Sedangkan dari segi waktu kompresi dan dekompresi, LAC masih membutuhkan waktu yang lama jika dibandingkan WavPack dan FLAC. Hal tersebut menyatakan bahwa algoritma pada WavPack dan FLAC masih lebih baik.

SIMPULAN

Dari hasil evaluasi didapatkan bahwa jenis musik (*music genre*) berpengaruh terhadapa rasio kompresi. Semakin lembut musik, akan menghasilkan rasio kompresi yang semakin baik, sebaliknya, semakin keras jenis musik, maka rasio kompresi akan semakin buruk. Hasil kompresi *Lossless Audio Codec* (LAC) belum optimal dan kecepatan kompresi dan dekompresi relatif lebih lambat dibandingkan dengan aplikasi komersial yang ada di pasaran, tetapi LAC mudah dikembangkan karena algoritma yang digunakan cukup sederhana.

DAFTAR PUSTAKA

McGloughlin, S. (2001). *Multimedia: Concepts and Practice*. New Jersey: Prentice Hall

Morken, K. (2007). *A Computational Perspective on Calculus.* Oslo: University of Oslo

Ashland, M. T. (2000). *A Fast and Powerful Audio Compressor.* Monkey's Audio

Whiters, W. D. (1996). *A Rapid Entropy Coding Algorithm.* Annapolis: United States Naval Academy